Model Development Using Machine Learning Models

September 18, 2023

Outline

- Tree Ensembles (review)
- Artificial Neural Networks
- Hyper-Parameter tuning

Last Time: Generalized Linear Model Framework

In the GLM framework:

- A distribution for the response Y is appropriately chosen
- A function of the conditional mean of the response is modeled as a linear combination of the predictors:
  - $g(E[Y|X]) = g(\mu) = \boldsymbol{X\beta} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$
- where $g(\cdot)$ is called the link function, and relates the linear combination of predictors to the mean of the distribution of y

- **Nonlinearity** and **interactions** need to be included with appropriately selected variable transformations

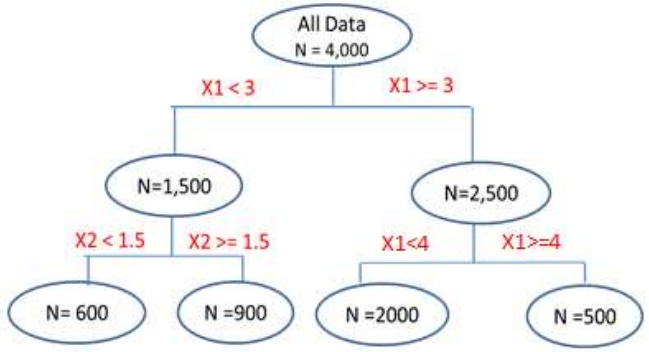
Generalized Additive Model (GAM) :

$$E(Y|X) or\ logodds(p) = f(X) = \alpha + f_1(X_1) + f_2(X_2) + \ \ldots + f_p(X_p)$$

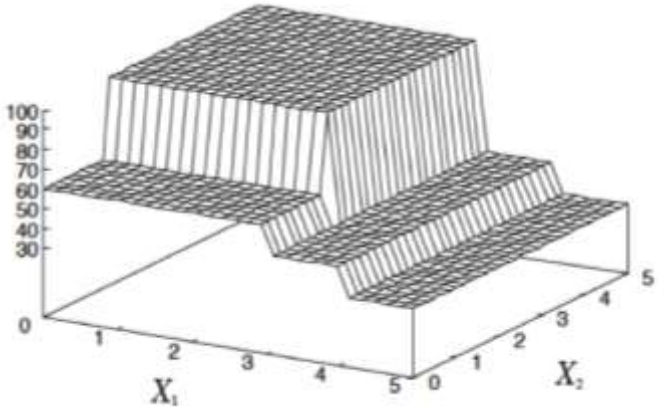
$f_j(X_j)$: unspecified smooth non-parametric functions

Tree Ensemble Review

Decision Trees

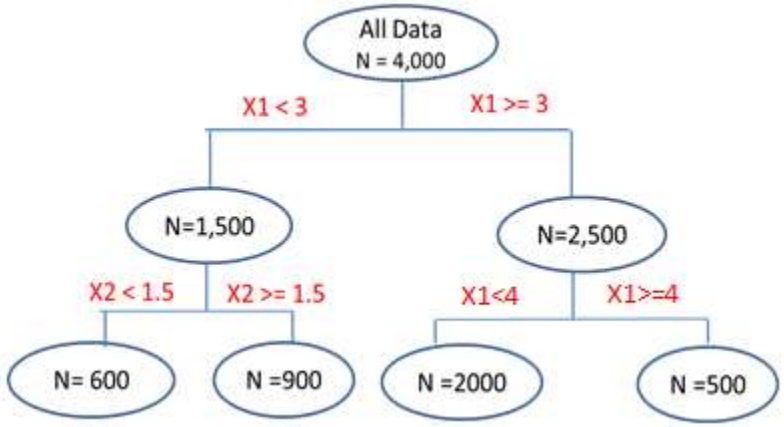


- Decision trees partition the feature space into a set of rectangles and fit a simple model (e.g., constant) in each one.
- Advantages:
  - Fast, intuitive
  - Able to handle both numeric and categorical data
  - Robust to outliers in predictors
  - Model interaction and nonlinearity automatically (little data transformation)
- Disadvantage:
  - High bias for shallow trees, for example trying to model linear relationships
  - Unstable, high variance for deep trees. Small change in data can result in a completely different tree



CART

- CART (**c**lassification **a**nd **r**egression **t**ree) by Breiman et. al.(1984).
  - Starts from the root node with all data
  - Splits into child nodes based on a variable → child node as homogeneous as possible.
    - Loss functions for measuring homogeneity:
      - MSE(continuous)
      - Gini-Index/cross-entropy/logloss (binary/multi-class)
  - Pruning: grow deep tree → prune to minimize cost complexity
  - Results can be viewed as a simple piecewise constant regression model
  - $\hat{Y} = c_1 \times I(X_1 < 3, X_2 < 1.5) + c_2 \times I(X_1 < 3, X_2 \geq 1.5) + c_3 \times I(3 \leq X_1 < 4) + c_4 \times I(X_1 \geq 4)$



- Model based trees – fit a simple model at each node instead of constant
- Algorithms with more than two splits (CHAID by Kass 1980)
- M5(Quinlan (1992)), LOTUS(Chan and Loh(2004)), SLIM(Hu et. Al. 2020)
- Implementations
  - Scikit-learn: DecisionTreeRegressor and DecisionTreeClassifier
  - R: rpart package
  - Spark: mllib library

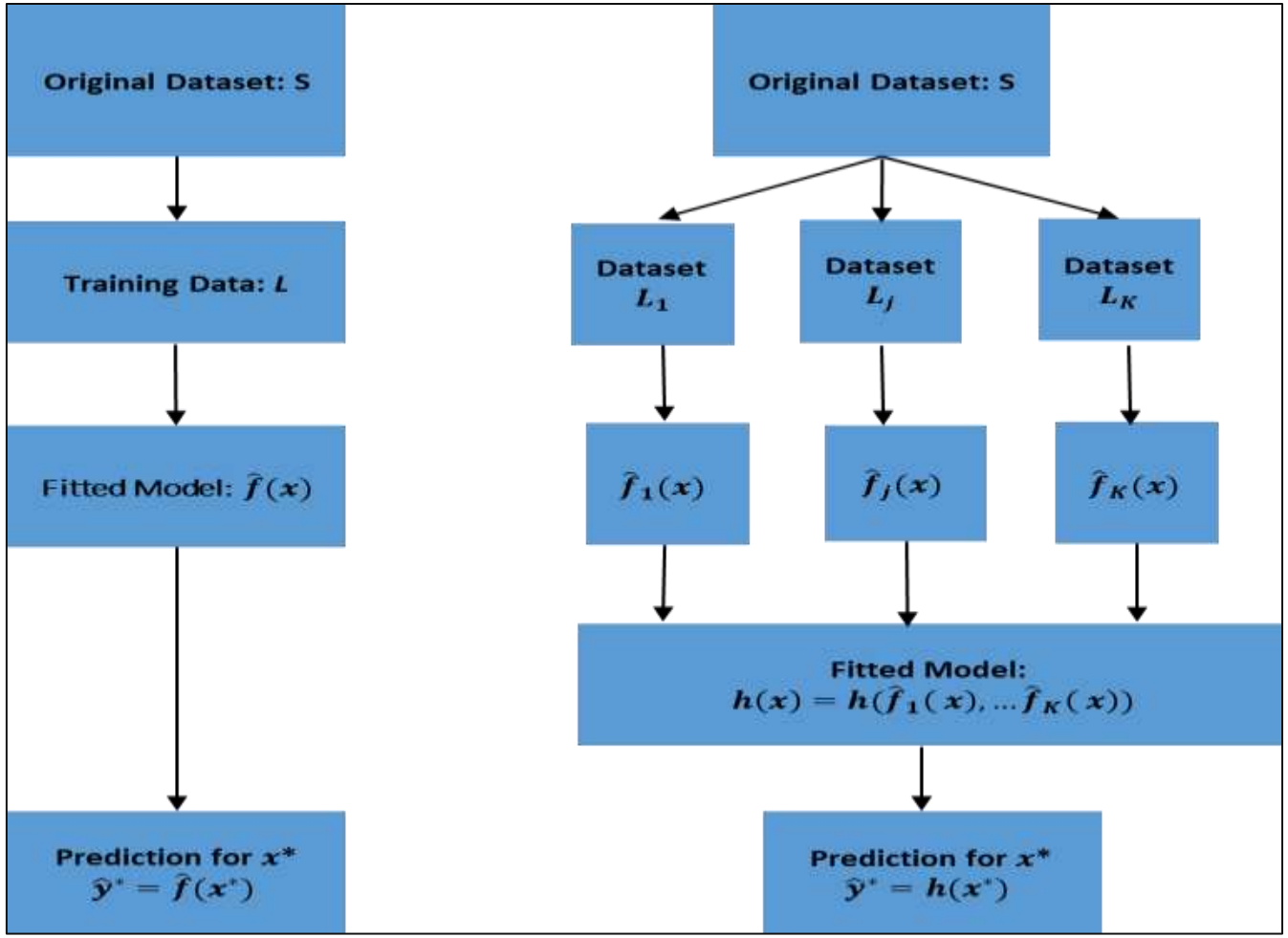
Performance comparison

- Using the UCI Bike Share data on rental counts.

	Model	test_MSE	test_MAE	test_R2	train_MSE	train_MAE	train_R2	Time
2	Tree	0.0052	0.0436	0.8511	0.0040	0.0383	0.8868	0.0
1	GAM	0.0114	0.0791	0.6762	0.0110	0.0778	0.6868	0.2
0	GLM	0.0221	0.1103	0.3727	0.0223	0.1122	0.3637	0.0

→ Able to capture **interactions** in the data and hence higher predictive performance

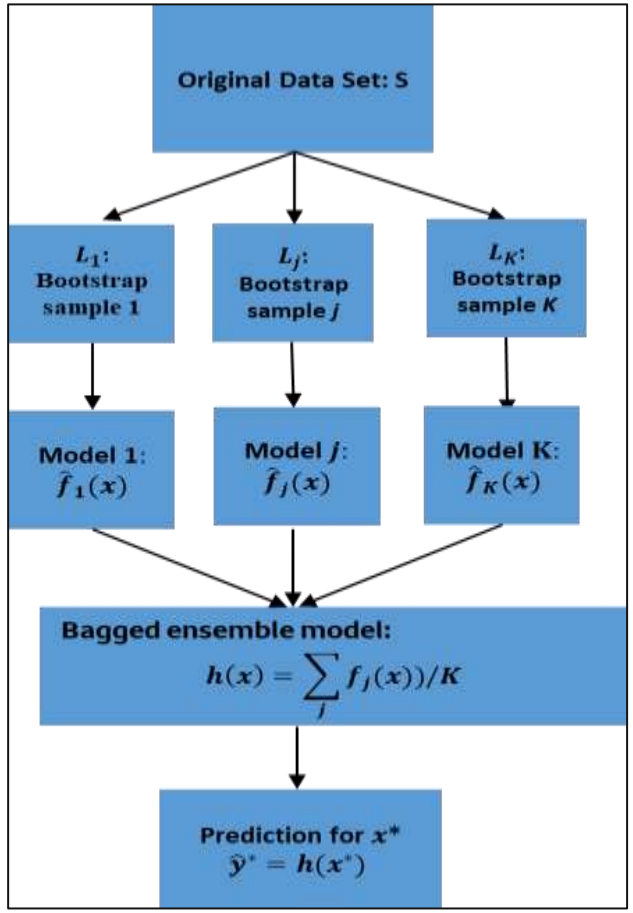
Ensemble algorithms



- Improve performance by combining the outputs of several individual learners.
- Examples:
  - Bagging
  - Boosting
  - Model Averaging
  - Majority Voting
  - Ensemble Stacking
- Mostly bagging and boosting algorithms use tree based learners.

Bagging

- Bagging: bootstrap aggregating(Breiman in 1994)
  - "improvements for unstable procedures" (Breiman 1996, example: deep decision tree
- Algorithmic framework:
  - bootstrap sample at each iteration $i, i = 1, 2, \ldots, n$.
  - Fit base learner to bootstrap sample → $\hat{f}_i(x)$
  - Combine base model predictions :
    - Averaging (Regression):$\hat{f}(x) = \frac{1}{n}\sum_i \hat{f}_i(x)$
    - Majority voting (Classification): $\hat{f}(x) = \arg\max_k \frac{1}{n}\sum_i I\left(\hat{f}_i(x) == k\right)$
- Loss functions:
  - Tree based learners grow trees using same loss functions as in CART
    - Loss functions for measuring homogeneity in leaves:
      - MSE(continuous)
      - Gini-Index/cross-entropy/logloss (binary/multi-class)
- Combining base model predictions reduces variance, making model stable
  - More base learners → better prediction, higher computational complexity
  - Base learners: low bias high variance



Why bagging works?

- Continuous response:
  - Assuming independence of base learners, averaging reduces variance
  - $Var\left(\frac{1}{n}\sum_i \hat{f}_i(x)\right) = \frac{\mathrm{Var}\left(\hat{f}_i(x)\right)}{n}$

- Binary response:
  - Assume Bayesian optimal decision at x is 1
  - $P\left(\hat{f}_i(x) = 1\right) = e > 0.5$
  - $\sum_i I\left(\hat{f}_i(x) = 1\right) \sim Binomial(n, e)$ [i.i.d base learners]
  - $P\left(\hat{f}(x) = 1\right) = P\left(\frac{1}{n}\sum_i I\left(\hat{f}_i(x) = 1\right) > 0.5\right) \to 1$ as $n \to \infty$

- In reality, base learners are not independent due to overlap in bootstrap samples
  - Variance $\to c \neq 0$ as n increases.
  - Correlation limits the reduction of variance. How to further de-correlate the base models?

Random Forests

- **Random Forests** (Breiman 2001)
  - Bagging and random feature subsampling
- Deep Trees
  - Low bias, high variance
  - Reduce variance through bagging
  - A variant uses sample without replacement
- De-correlate trees
  - use random subset of features in each split instead of entire feature set
  - Tries to achieve maximum variance reduction
- Typically over-fits the data
- Typical Hyper-Parameters (HP) to be tuned (sklearn)
  - N_estimators : number of forests
  - Max_depth : maximum depth of the trees
  - Min_samples_leaf (MSL): the minimum samples required at each leaf
  - Max_features: number of features to consider at each split

Boosting

- Boosting is a different type of ensemble algorithm, based on removing bias of a simple learner.
- Given a simple learner, can you improve it to be a strong learner? (Kearns and Valiant 1988)
- Schapire (1989): Yes → by a technique called “boosting”
- Freund and Schapire (1995): AdaBoost for classification

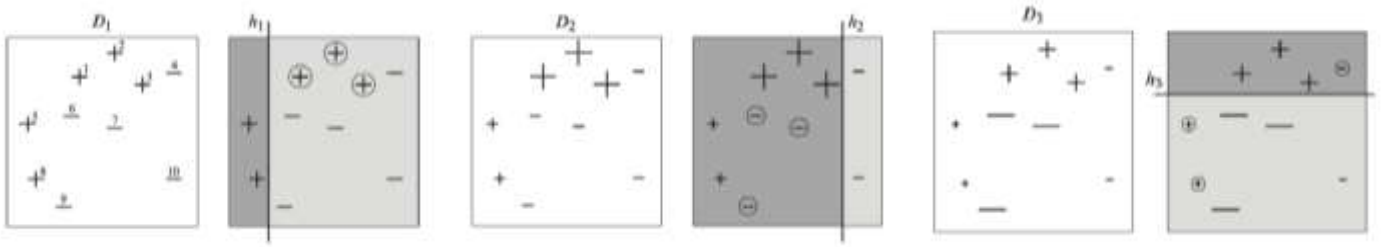
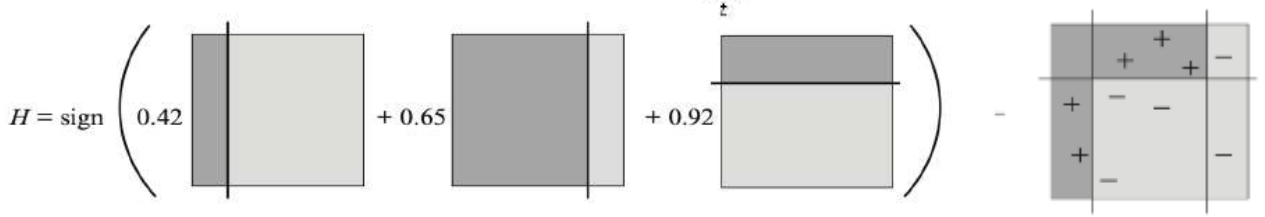


Figure : AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

- “Base learner”: simple rectangular classification regions at each stage
- Reweighting at each stage – more weight to data that are misclassified
- Fit an additive model (ensemble)

$$H(x) = \sum_t \rho_t h_t(x)$$



Source: Figure 1.2 of [Schapire and Freund, 2012]

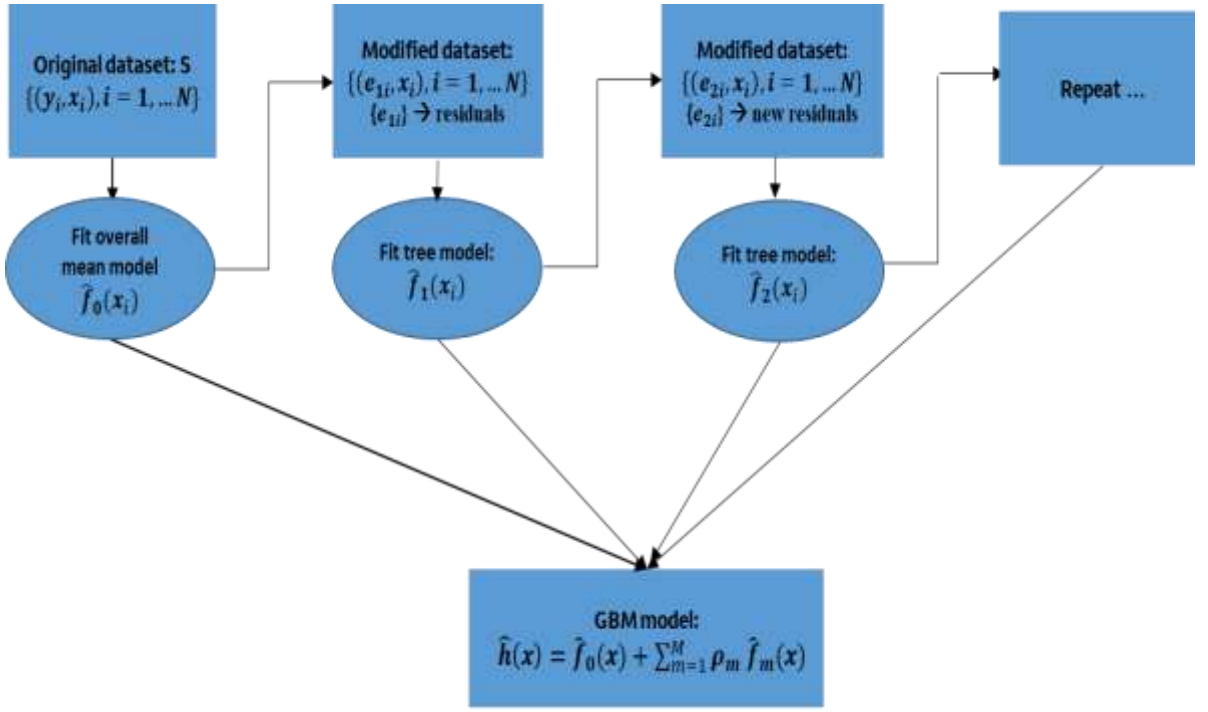
Gradient Boosting

- Breiman (1998+): Boosting → optimization algorithm
- Friedman (2000+): Extended concept to gradient boosting (gradient descent)
- Loss function to minimize: $L(y, f)$.
  - squared error loss $(y - f)^2$ for regression
  - deviance $yf - \log\left(1 + e^f\right)$ for binary classification (f is the logodds) (deviance = logloss)
  - Other loss functions: absolute error loss, partial likelihood, etc
- Find the prediction function $f(x)$ that minimize the total loss $\sum_{i=1}^{N} L\left(y_i,\ f(x_i)\right)$.
  - $f(x)$ is optimized in an additive, stage-wise way: $f(x) = T_0(x) + \sum_{m=1}^{M} \eta_m T_m(x)$, where $T_0(x)$ is baseline (e.g., overall mean in regression).
  - In each stage m, update $f(x)$ in the direction $T_m(x)$ where the total loss decreases, for a step size/learn rate of η_m.
  - Each base learner $T_m(x_i)$ is fit to the negative gradient (gradient descent) from the previous iteration
  - $T_m(x_i) = -\frac{\partial L(x_i, y_i)}{\partial\, \hat{f}_{m-1}(x_i)}$

Gradient Boosting

- Stochastic gradient boosting (Friedman 1999): fit each tree with a subsample instead of the entire data. This can be more robust and less overfitting.
- Hyper-Parameters (HP):
  - number of trees,
  - learn rate, tree
  - depth, …
- Implementation:
  - Scikit learn: GradientBoostingClassifier and GradientBoostingRegressor
  - R: gbm package
  - Spark: mllib library
  - H2o: h2o.gbm
- Other popular variations
  - **XGBoost**
  - **LightGBM**
  - CatBoost

Illustration of GBM for continuous regression



XGBoost

- XGBoost (Extreme Gradient Boosting) (Chen and Guestrin 2016) - a variant of GBM that uses second order Hessian
- XGBoost adds a **penalty** term to the loss function

$$\sum_{i=1}^{N} L\big(y_i, f(x_i)\big) + \sum_{m=1}^{M} \Omega\big(T_m(x)\big)$$

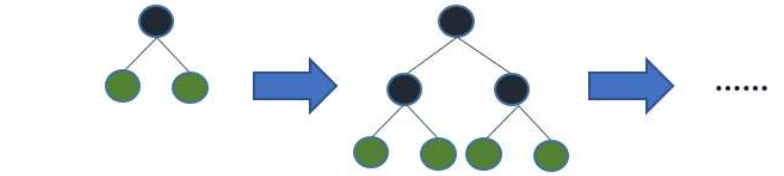
  to control overfitting.
  - $\Omega(T) = \gamma|T| + \frac{1}{2}\lambda \sum_j w_j^2$, where $|T|$ is the number of terminal nodes in the tree and w_j is the fit in terminal node j.
  - The parameter γ is a pruning parameter: any split with the improvement below γ is pruned.
  - The parameter λ acts like a shrinkage parameter: the prediction in each tree node is shrunk $\mathrm{w_j} = -\frac{G_j}{H_j+\lambda}$, where $G_j = \sum_{i\in I_j} g_i$, $H_j = \sum_{i\in I_j} h_i$ are the total gradients and total **Hessian** in Node j.
  - An L1 penalty ($\alpha \sum_j |w_j|$) can be added as well.
  - The trees are built **depth-wise**.
- XGBoost has the following advantages:
  - Parallelized. So it is scalable.
  - Penalized, to reduce overfitting issue.
- It was originally implemented in C++ but it is available in Java, Python, R through APIs. Has Scikit-learn wrapper.
- Hyper-Parameters(HP) include : max_depth, learning_rate (lr_rate),N_estimators (# trees), L1 (alpha),L2 (lambda), min_child_weight, …

LightGBM

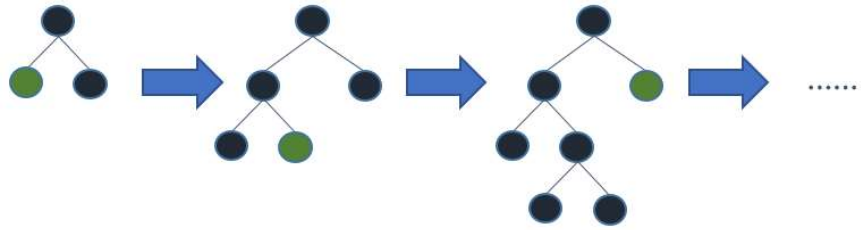
- Introduces two new concepts (Ke et al, 2017)
  - Gradient One-Sided sampling(GOSS)
    - At each iteration, keep all observations with large gradients and random subset on instances with smaller gradients.
  - Exclusive feature bundling (EFB)
    - Bundles up mutually exclusive features.
    - Helpful in feature reduction for high dimensional data with large number of 0-1 encoded columns.
- Builds trees **leaf-wise**
- Hyper-Parameters:
  - Lr_rate
  - Max_depth
  - Num_leaves
  - Min_data_in_leaf
- Developed by Microsoft. Built in C++, has python and R interface.

XGBoost vs LightGBM

- Numerous blogs on comparison
- Similar performance
- LightGBM is faster
- XGB by default grows tree level-wise(split node closer to root)
- LightGBM grows trees leaf-wise
- Difference in the algorithm may lead to different feature importance and other diagnostics.



Level-wise tree growth



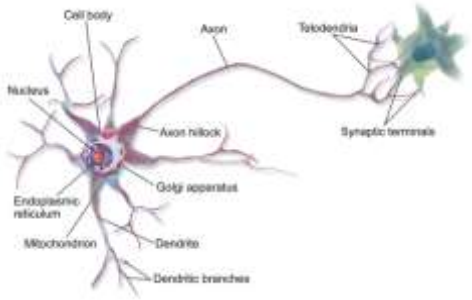
Leaf-wise tree growth

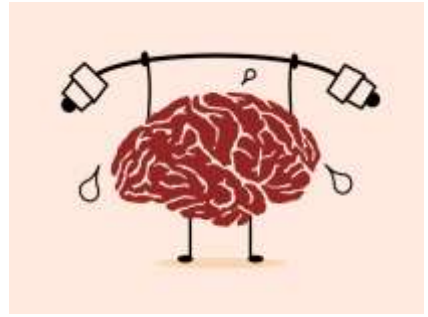
Artificial Neural Networks

Neural Networks: history and inspiration from neuroscience

Wave 1: “Cybernetics”: [1940s-1960s]

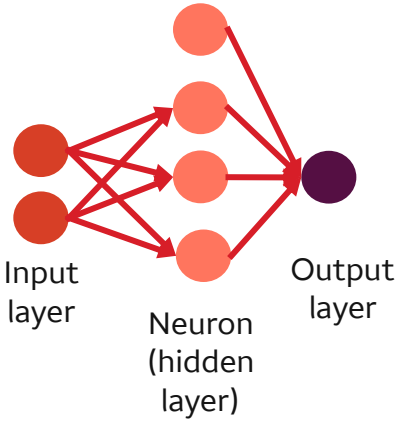
- Model of biological function of brain
- 1958 Rosenblatt: Developed “perceptron”, a training algorithm
- Effectively ended by *Perceptrons* (Minsky and Papert), who showed difficulties with current approaches.

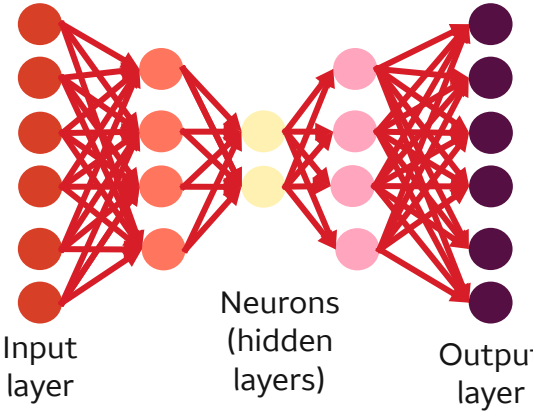




Wave 2: “Connectionism”: [1980s-1990s]

- From the study of how the brain can form connections.
- Central Idea: A network of many simple units can learn complex patterns.
- The backpropagation algorithm provided an essential advance in training neural networks.





Third Wave [2006-present]: “Deep Learning”

- Ability to train neural networks with more layers.
- Able to outperform other ML systems:
  - Object recognition in pictures
  - Natural Language Processing

Feedforward Neural Networks (FFNN):

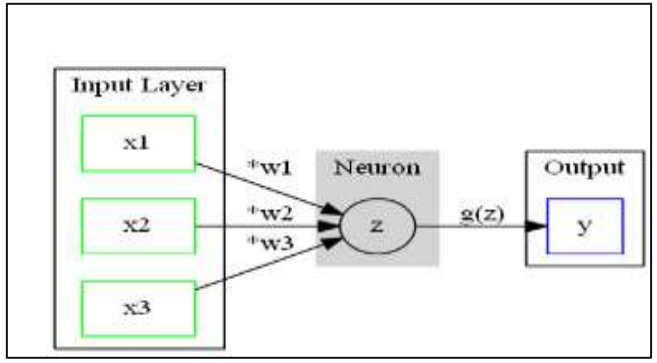
- Activation function: $g(w^T x)$
  - Sigmoidal, Hyperbolic Tan, ReLU
  - Connection to **additive index models:**

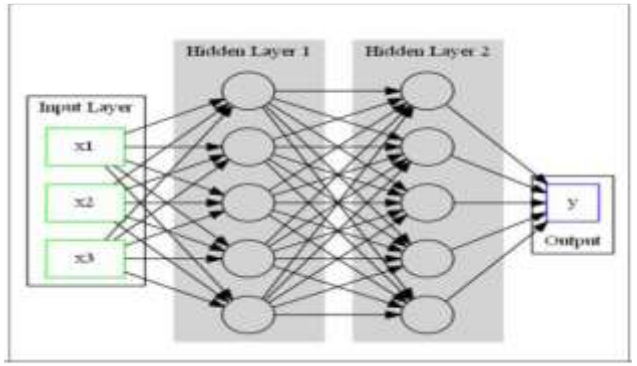
$$f(\boldsymbol{x}) = g(w_1 x_1 + \ \ldots + w_P x_P)$$

- FFNN architecture
  - Nodes (Neurons)
  - Input, Output, and Hidden Layers
  - All nodes connected with others in next layer

- Hyper-Parameters(HP)
  - Learning rate
  - Number of layers
  - Neurons in each layer
  - L1, L2, dropout
  - batch-size
  - activation function…

- Implementation
  - Scikit-learn – MLP
  - Keras
  - PyTorch





Single Layer

- "Universal Approximation Theorem":
  Wide hidden layer +squashing activation function
  can approximate any well behaved function arbitrarily well
- Possible over-fitting

Multi-Layer

- Increased representation power
- Less prone to over-fitting
- Increase in computation cost and lack of interpretability

Fitting a Neural Network to Data: Learning the Weights

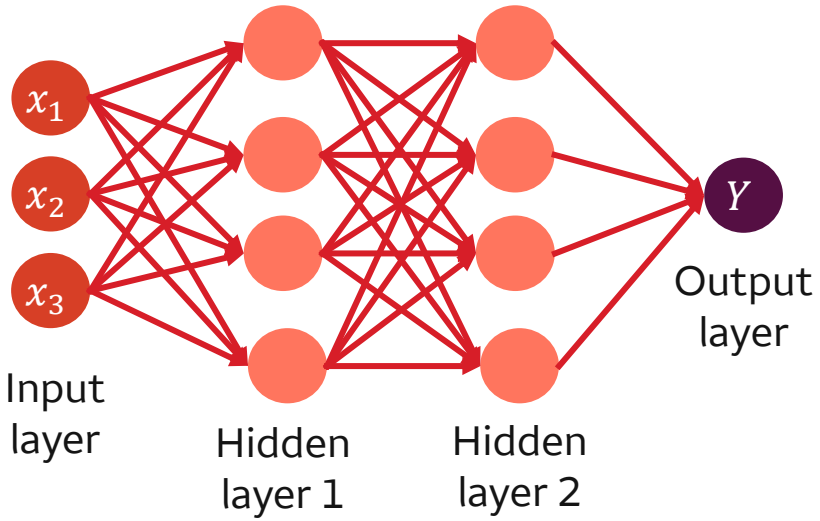
- The **weights** (and **bias**) of each neuron are the unknown parameters in an ANN that need to be learned from data.
- To do so, we define an appropriate cost function that:
  - Represents an average of the cost of individual observations in the training set.
  - Should be a function of the outputs from the ANN and the response y.
  - Examples:
    - For continuous responses, we use squared error loss: $\frac{1}{n}\sum_i(y_i - \hat{y}_i)^2$.
    - For binary response, we use cross entropy, or log loss: $-\frac{1}{n}\sum_i[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$
    - For multinomial responses, we use a generalization of cross-entropy, with *j* indexing category:

$$-\frac{1}{n}\sum_i\sum_j\left[y_i^{(j)}\log\hat{y}_i^{(j)}\right]$$

- Choose the **weights** and **biases** that minimize the cost function.
  - In principle, this can be achieved using calculus.
  - However, many of these solutions are not easily implemented in ANNs.

Optimization: Back Propagation Algorithm

- Gradient Descent, can be challenging in NNs, due to computation of gradient.
- Preparation: Input the data x, and initialize all weights in the network.
- In practice, optimization of ANNs has become a research area:
  - many other algorithms and research (Adam, SGD, etc)
  - mini-batch training



Feedforward
- Feed the data through the network
- Compute the output of each node based on the current weights

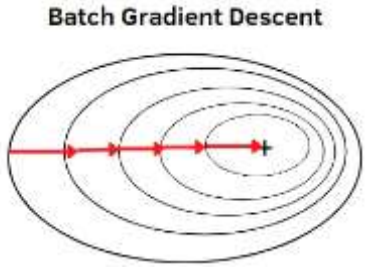
Gradient
- Compute the gradient of the cost function with respect to the last hidden layer

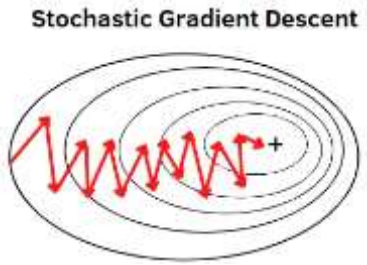
Backward propagation
- Work backwards through the network
- Compute the gradient of the cost function as they depend on the weights in the lower layers

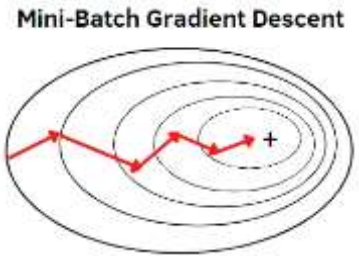
Update
- Update the weights using gradient descent
- Return to step 1 "Feedforward"

Different forms of gradient descent

- Batch gradient descent
  - computes gradients of all samples at each iteration
- Stochastic gradient descent
  - computes gradient of single random sample at each iteration
- Mini batch gradient descent
  - partitions data and computes gradient of one partition at each iteration







- Variations include
  - Adaptive learning rates
  - Adding momentum
- Different variations of gradient descent
  - Adagrad
  - RMSProp
  - Adam

Using ANNs in Practice

General Usage

- Very flexible. Choices for:
  - Structure: Number of Hidden Layers, Number of nodes on Each Hidden Layer, Activation Functions for each Hidden Layer
  - Regularization Strategy/ Parameters
  - Additional Features: Skip connections, Batch Normalization, Dropout, Constraints
  - Training/Optimization Algorithms
- Can make grid search difficult
- **Note:** Much of the literature available gives advice in the context of unstructured data (images/text/speech).

  Such advice may not always be useful in banking problems.

Training effectively

- Training can be challenging; saddle points and local minima can result in a sub-optimal model.
- Tips:
  - **Standardize** or **Normalize** data (X) before training. Avoids vanishing gradient problem.
    - Min/Max scaling
    - Gaussian standardization (perform better with large outliers)
  - **Batch Normalize** hidden layers.
  - optimization routine with **learning rate decay** (e.g. Adam).
  - **batch size** used in training - smaller batches can be slow and volatile, but help escape local minima/saddle points.
  - Use **early stopping** to determine number of training epochs.

Overfitting

- ANNs are flexible models, with a (potentially) large number of parameters, therefore overfitting is a concern.
- Strategies to avoid overfitting in ANNs include:
  - Multiple narrow layers vs. Single wide layer
  - Weight Regularization: Penalizing large weights in the cost function.
  - Dropout: Randomly set a fraction of neurons' activations to 0 during training. This forces redundancy of neurons, and reduces neuron specialization.
  - Early Stopping via a validation set.

Comparison of ML Algorithms

Below is comparison among some most popular machine learning algorithms.

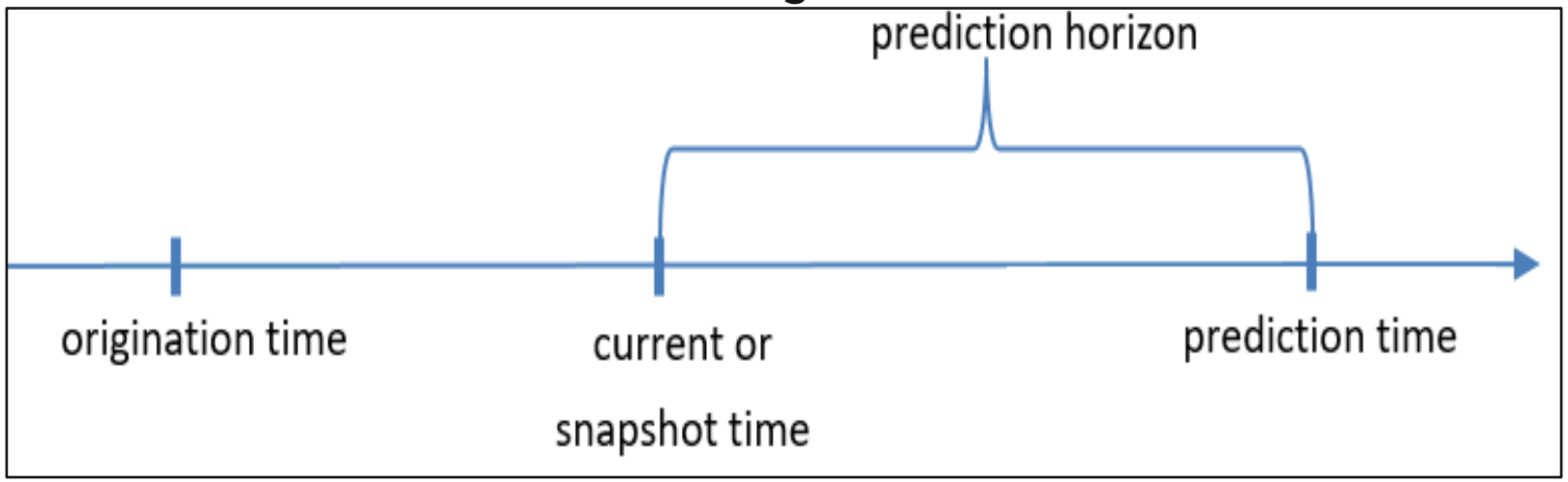
	Preprocess	Robustness to outliers in input space	Computational scalability (large N)	Predictive power	Smoothness of response surface	Feature engineering	Hyper parameter tuning
GBM	No, some require dummy coding	Yes, tree based methods are robust to outliers	Depend on software implementation. XG B, LightGBM and H2O GBM are scalable	Good, often outperforms random forest and neural network in prediction	The response surface for tree based methods are often jumpy and not smooth, especially for small data	Good for manually created features; may not be good for raw features, e.g., transaction data, image data.	Relatively easy
Random Forest	No, some require dummy coding	Yes, tree based methods are robust to outliers	Depend on software implement ation. Scikit learn random forest, H2O random forest are scalable	Good	The response surface for tree based methods are often jumpy and not smooth, especially for small data	Good for manually created features; may not be good for raw features, e.g., transaction data, image data.	Relatively easy
Neural Network	dummy coding and standardizatio n	No	Large neural network with large data requires GPU. Computation with simple network or small data is scalable on CPU.	Good. Best for image, speech,…	Usually smooth	Powerful in feature engineering with a variety of deep neural network structures.	Complicated

Other ML algorithms – knn, svm, … (Not scalable)

Application to Home Mortgage: Modeling "In-Trouble" Loans

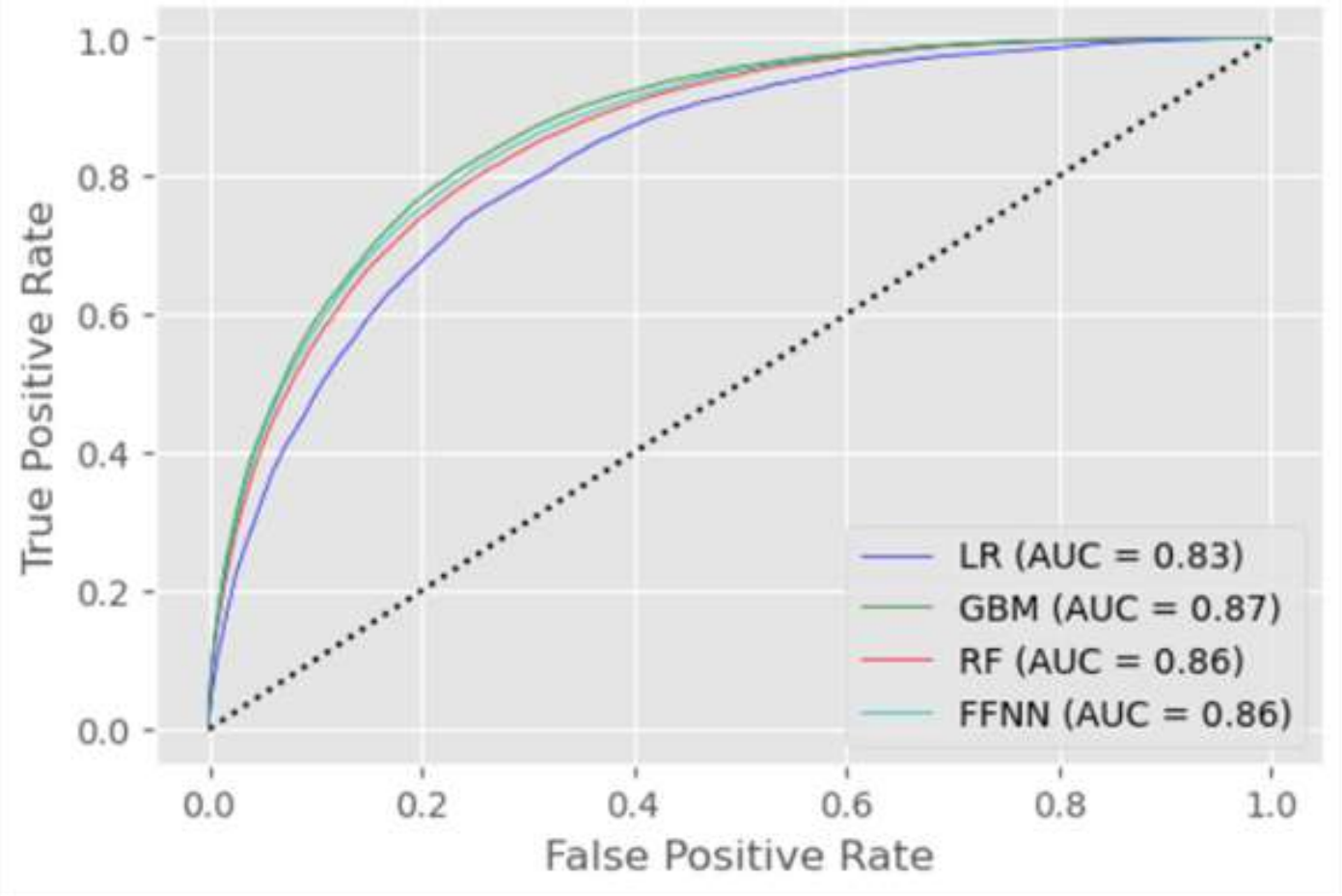
- One portfolio: ~ 5 million observations
- Response: binary = loan is "in trouble" (multiple failures and connections to competing risks)
- 20+ predictors: credit history, type of loan, loan amount, loan age, loan-to-value ratios, interest rates at origination and current, loan payments up-to-date, etc. (origination and over time)

Modeling framework



Loan origination, current (snapshot) and prediction times

Comparison of Predictive Performance: ROC and AUC on Test Data



- ML with 22 predictors
- LR model: eight carefully selected variables
  - snapshot fico (credit history);
  - ltv (loan-to-value ratio);
  - ind_financial-crisis;
  - pred_unemp_rate;
  - two delinquency status variables;
  - horizon

How typical is this "lift" in our applications?

Findings from internal study -XGB and FFNN are competitive and exhibit better model performance across a variety of functional forms than RF

https://arxiv.org/ftp/arxiv/papers/2204/2204.12868.pdf

Neural Network – other architectures

Complex Architectures

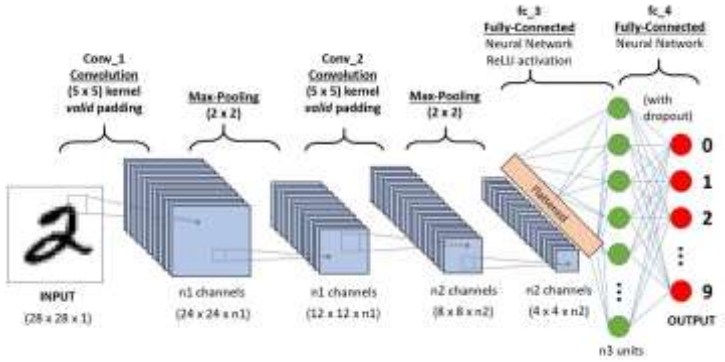
Convolutional NN

Used in images, text, time series

- Key Features:
  - Convolutional layers, where inputs are convolved with their neighbors.
  - Each output is a weighted average of the inputs:

$$\sum_{i,j} a_{i,j}\, x_{i,j}$$

  - The **weights** remain **constant** as the convolution is applied to successive windows of data.
  - Other tricks, like Pooling

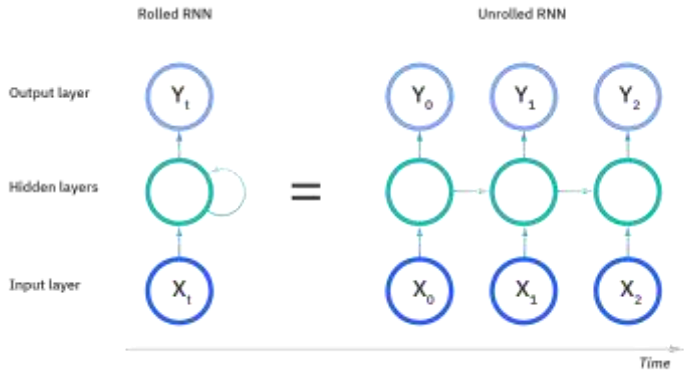


A CNN sequence to classify handwritten digits

a-comprehensive-guide-to-convolutional-neural-networks

Recurrent NNs

- Useful in studying sequences, such as in natural language context.
- Defined by “recurrent” connections, where the output of a downstream unit serves as input to an upstream neuron.
- Several variations; “Long Short-Term Memory” (LTSM) was popular; now focus on Attention Networks.

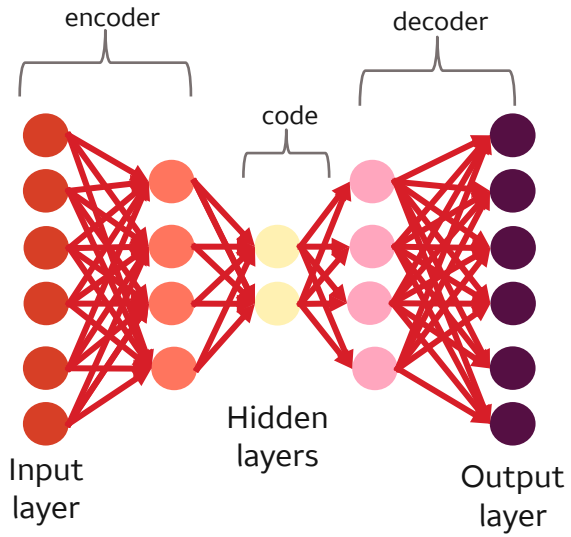


https://www.ibm.com/cloud/learn/recurrent-neural-networks

Complex Architectures 2

Auto-Encoders

- Dimension Reduction Network
- Predicts input from input.
- Bottleneck layer engineers lower-dimensional features.
- Different types
  - De-noising AE
  - Sparse AE
  - Variational AE …

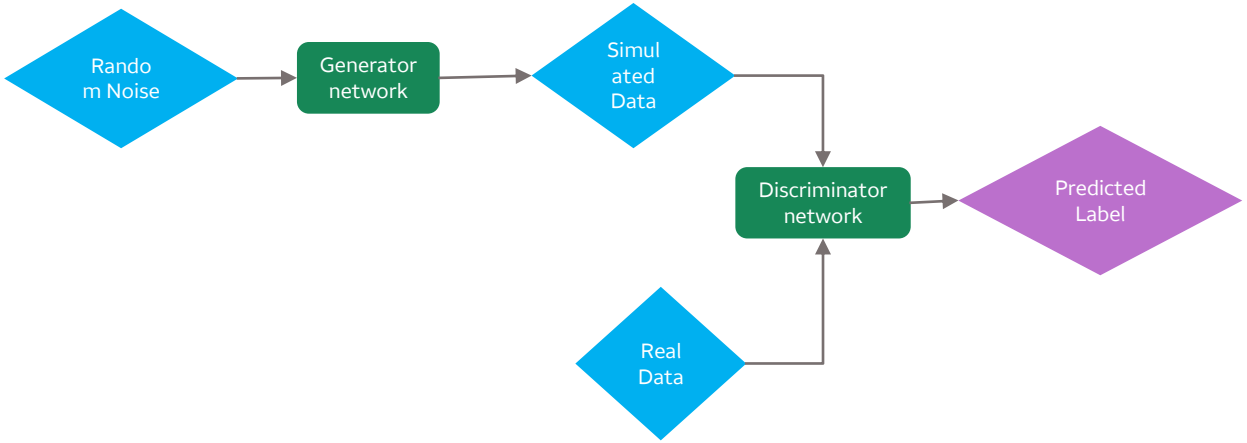


Guide to AE

GAN: Generative Adversarial Networks

Unsupervised technique

- Pair of ANNs, trained with simultaneous backpropagation
- A Generator Network, which produces candidate data examples
- A Discriminator Network, which learns to distinguish the generated data from the real data. (Classification)
- Simultaneous training improves the performance of both networks.



Advanced Architectures and Ongoing Research

- Attention Networks
  - Added mechanisms to model dependencies across data regardless of positions.
  - Originally proposed as a component of RNN structures; now used independently.
  - Reference “Attention Is All You Need” https://arxiv.org/pdf/1706.03762.pdf
- Temporal Fusion Transformers
  - Use a combination of RNN and Attention structures to predict multi-horizon time series
  - Provides some measure of interpretability into temporal patterns
  - Reference: “Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting” https://arxiv.org/pdf/1912.09363.pdf
- Other exciting work, both in the bank and in the broader research community

Pros and Cons of using Neural Network Algorithms

Pros

- Flexibility: ANNs can be use in a variety of machine learning tasks:
  - Regression
  - Classification
  - Dimension Reduction/ Feature Engineering
  - Text Processing
- Feature Engineering: ANNs show potential to extract meaningful features from raw data.
- Exploiting Large Data: ANNs have been observed to continue receiving performance gains from larger training data sizes when other algorithms reach a plateau.
- Flexibility in Inputs: ANNs are capable using a variety of data types, as well as combining differing data types.

Cons

- The “black box” problem: Often difficult/impossible to explain *why* the network makes the predictions it does.
- Computational Cost: Training ANNs can have a high computational cost, and may not be efficient when another tool (e.g. Gradient Boosting) will perform well.

Software

- Specialized software packages exploit computational graphs, symbolic differentiation, back propagation and mini-batch training.
- Popular Packages include:
  - **TensorFlow /Keras**(Google)
  - **PyTorch**/Torch
  - Caffe/Caffe2
  - CNTK (Cognitive Toolkit) (Microsoft)
  - Theano
- Wrappers exist to provide abstraction layers (Keras)
- Many general-purpose scientific packages have more limited implementations (R, MatLAB, scikit-learn)
- Many offer distributed or flexible computing
- NVidia providing growing support for GPU computation

Additional Considerations: Monotonicity and Reproducibility

Monotone constraints on ML algorithms

- Monotonicity is an important model requirement in some applications like credit scoring. Customer with better credit, longer history, less inquires should be more likely to be approved.

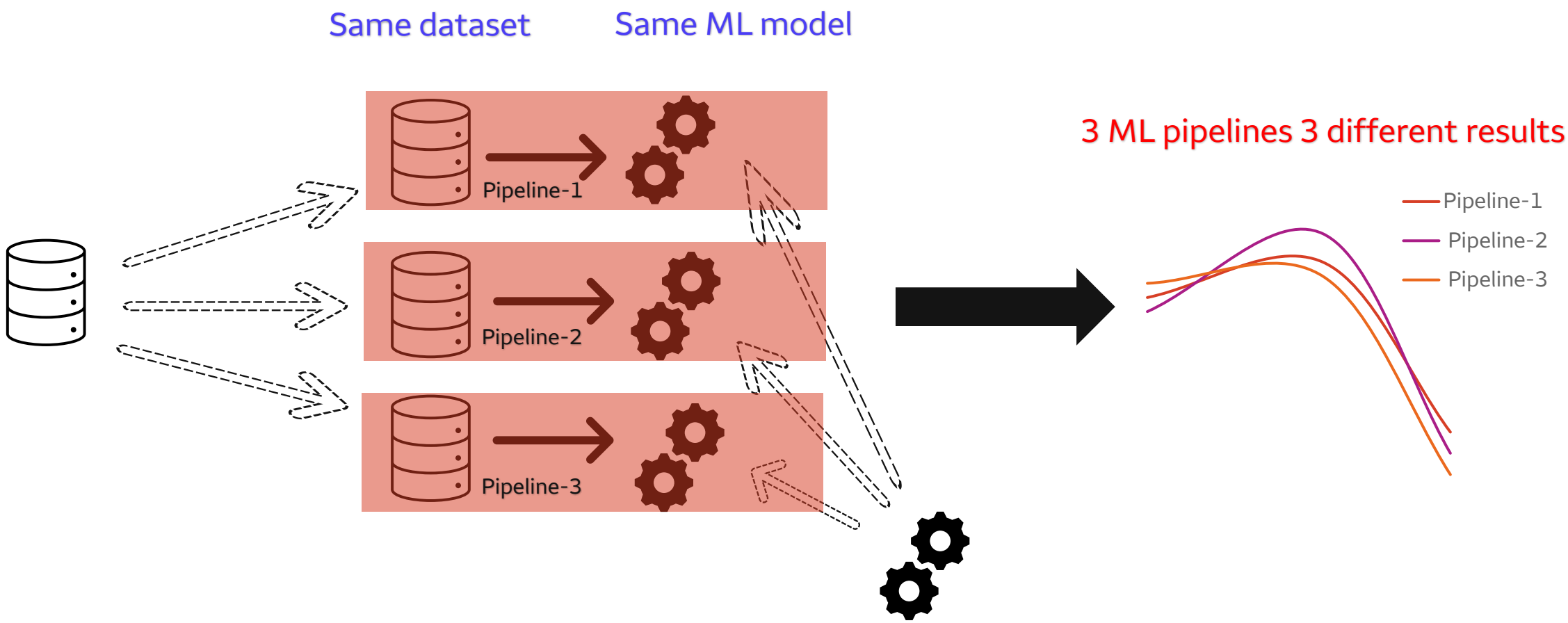
- A variable x_j is said to be monotone increasing, if $f(x_j = a, \boldsymbol{x}_{-j}) \leq f(x_j = b, \boldsymbol{x}_{-j})$ for any $a \leq b$ and $\boldsymbol{x}_{-j}$. Similarly, monotone decreasing can be defined.

- Different monotone ML algorithms are available:

  - Soft monotone ML algorithms. These algorithms add a penalty on the loss function, eg,
    - **KiGB** (knowledge-intensive gradient boosting) algorithm: it penalizes the difference between average of all leaves in left vs right subtree. This method is flawed since mono on average is a weaker requirement.
    - **Certified monotonic neural networks** (Liu et al. 2020) calculates the gradient of a nnet and adds a penalty if the gradient violates the monotonicity. If violations are found, the penalty parameter λ is increased and the model is retrained again until no violations can be found. But in practice, it is NP hard to find all violations.

  - Hard monotone ML algorithms: these algorithms are 100% monotone by design
    - **XGBoost** imposes split-wise and branch-wise limit to achieve monotonicity. It works well for shallow trees
    - **Isotonic classification tree RF** (Bonakdarpour et al. 2018): fit a conventional RF, then use isotonic regression to update leaf values so monotonicity is satisfied.
    - **Tensorflow Lattice** (Gupta et al. 2016): use Lattice with monotonicity constraints.

A Note on Reproducibility in Machine learning



Many Machine learning models (Xgboost, Random forest, DNN, etc) have dependence on parameters that are randomized

Randomization exists at different levels in ML

Dataset	Sources of randomness
	-*Train/Test/Validation split* -*Cross Validation split*
❑ Split is randomly chosen and depends on a random seed. ❑ Different training splits will generate a different model and hence difference in results.	

Model	Sources of randomness controlled by model random seed
RF	• Bootstrapping • Row subsampling • Column subsampling
GBM	• Row subsampling • Column subsampling • Selection of validation set while early stopping
XGB	• Row subsampling • Column subsampling • Selection of validation set while early stopping

Summary

- We talked about the different machine learning models, focusing on supervised machine learning methods like random forest, GBM and neural network.
- Random forests and GBM are tree based ensemble methods, designed to improve the performance of a single tree using a collective set of trees. Neural network is a biologically inspired method designed to mimic the function of brain.
- In structured data XGB and FFNN are shown to have competitive performance and both outperform RF.
- Machine learning algorithm can achieve better performance than traditional statistical regression, without manual variable selection or transformation.
- Hyper-Parameter tuning is an essential component of training Machine Learning models
- Reproducibility is important to replicate results as claimed and hence, it is important to understand the sources of randomness in machine learning modeling.

References

- Breiman, L. (1996). Bagging predictors. Machine Learning, 24 (2): 123–140.
- Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1):81-106.
- Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
- Friedman, J. H (2001). Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics, 29 (5): 1189-1232.
- Friedman, J. H. (1999). *Stochastic gradient boosting.* Stanford University.
- Hastie, T.; Tibshirani, R.; Friedman, J. H. (2001). *The elements of statistical learning : Data mining, inference, and prediction.* New York: Springer Verlag.
- Breiman, Leo (2001). Random Forests. Machine Learning, **45** (1)
- Deep Learning, by I. Goodfellow, Y. Bengio, and A. Courville, available: http://www.deeplearningbook.org/
- Explained: Neural networks, by L. Hardesty, available: http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
- M Nielsen (2017), Neural Network and Deep Learning online book: http://neuralnetworksanddeeplearning.com/index.html
- Goldstein, A., Kapelner, A., Bleich, J., & Pitkin, E. (2013). Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *eprint arXiv:1309.6392*.
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should I trust you?: Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 1135-1144)
- Hu, L et. al (2022). Surrogate locally-interpretable models with supervised machine learning algorithms, Journal of Indian Statistical Association.
- Chen, J. et al (2020). Adaptive Explainable Neural Networks, arXiv:2004.02352
- Vaughan, J. et al (2018). Explainable Neural Networks based on Additive Index Models. The RMA Journal